ATOMIC CONSTRATIONS

L. D. Marks, Updated November 2020

Contents

Introduction

With this version of the mixer (**10.1**, **Mango+ and later)**, a number of linear constraints on the atomic positions have been included. These allow for a large number of different ways to manipulate structures ranging from calculation of Transition States to other, symmetry constraints and (perhaps) even for phonon or Berry phase calculations. These notes will first describe how to use them, then more background and options. This document, and the code are beta+ versions.

Note that with the conversion to a full internal control of parameters with version 10.6.9 (Fall 2020) the functioning of these constraints has changed. The recommended and default version is now **WALK** with a bound of 20 mau per step (via **TRUST**).

Use

The code requires a file **case.constraint** to provide information. It will then blindly obey what is present, so a little care is needed. (In the future scripts could be generated to automatically generate these files for different types of problems.) All the constraints are of the form **Ax=b**, where **A** is a matrix for each constraint, **x** are the non-equivalent atomic positions and **b** is a scalar. The code calculates the minimum-norm shift of the atomic positions that would satisfy the constraints, then adds this to the pseudo-forces multiplied by a scalar which is either fixed or automatically. A full density and atomic positions optimization is then performed, with the end result close to obeying the constraints.

Three forms for the input are available. The first two are "simple" constraints which are specified by the matrix **A**, the third has the equivalent form that corresponds to displacing one or more atoms relative to a reference structure.

The different modes are switched based upon the first line of **case.constraint** :

NUMEQ SCALE FCONST Number of constraints (integer) and User Scale
then constraint strength Free Format

If **NUMEQ** is less than 100 the complete matrix **A** is needed (free format), for instance
AX1 AY1 AZ1 x, y, z components for atom 1 of the matrix **A**
AX2 AY2 AZ2 x, y, z components for atom 2 of the matrix **A**
...
b value of **b**

for each constraint.

If **NUMEQ** is more than 100, 100 (or a multiple of it) is subtracted and a Sparse input is used for each constraint, for instance to constrain the combined positions of atoms 1 and 4:

NVALS Entries for this constraint (integer)
ATOM1 AX1 AY1 AZ1 Integer for atom 1, then the three matrix values

ATOM4 AX4 AY4 AZ4	Integer for atom 4, then the three matrix values
b	Value of the constant **b** for this constraint

As an example, an input file **case.constraint** of form

201 1 0.1
1
3 0 0 1
0.589

will apply a constraint that the z co-ordinate of atom 3 is 0.589. A second example is a file

201 1 0.1
1
3 0 1 -1
0.0

Which will optimize the atomic positions with the constraint that the sum of the **y** and **z** coordinates of atom 3 are the same.

If **NUMEQ** > 1000, the calculation is referenced to an existing set of atomic positions and a vector displacements. The format is

Fraction	Free Format, fraction to move along the vector
Number_Vector_1	Integer, free format, the number of components
Filename	Path of the reference positions
Number_Pos_1	Integer, free format, number of vector displacements
N1 X1 Y1 Z1	Atom **N1**, and components **X1, Y1, Z1** of vector.
	This is repeated **Number_Vector_1** time

As a specific example, the file:

1002 1 0.1	**NUMEQ SCALE, FCONST**
0.125	Fraction to Move
/home/ldm/Mixing/NEB/Model3c/Struct1a/Struct1a.struct	
1	Number of atoms to constraint
1 0.0 -0.5 0.0	Vector displacement for this atom

will take the positions from Struct1a, and perform a constrained optimization forcing a displacement of the y co-ordinate of atom 1 to move 0.125 along the vector [0,-0.5,0].

Under The Hood

If you just want to get started skip this section. However, you may be confused so reading it is recommended, and also the later section on corrections.

Earlier version of Wien2k (and other DFT codes) handle constraints by fixing atomic positions. While this can work it has some mathematical problems. Fixing a position is equivalent to minimizing a problem over all atomic positions with an additional infinitely large penalty function for how much one or more of these deviates from the constraint. It is know that large penalty functions are ill-conditioned, and may get stuck in local minima or not converge. An alternative approach is to use Lagrange multipliers, but then one has to work out what value to use for these. The idea is to apply an extra fictitious force on target atoms to move them towards the desired final position.

In the first version of this code a number of different Lagrangian methods were used. With the transition to a more complete algorithm control of parameters in version **10.6.9** a much simpler approach now appears to be significantly better. This applies a fictitious force with a bound set by a **TRUST** region control for how much movement there should be (default 20 mau per iteration). If there is not enough movement towards the desired endpoint the force is increased; if there is too much it is decreased. This mode, **WALK** is now the default.

A standard alternative is to use what is called an Augmented Lagrangian method which combines Lagrange multipliers with a much weaker penalty function. (Both standard Lagrangian and Augmented are implemented.) These are well established methods for updating the Lagrange multiplier, and it is known that the method converges for relatively small penalty functions. This therefore allows the algorithm to escape local problems and converge much better – it is much better conditioned. (As a caveat, it may take too large a step.) The standard approach for minimizing, for instance, some function *A(x)* with a constraint *C(x)=0*, is to minimize the Lagrangian

$$L(x) = A(x) - \lambda C(x) + \eta/2 C(x)^2 \qquad (1)$$

where λ is the Lagrange multiplier and η is the strength of the penalty function, here a quadratic form. At each step during minimizing *L(x)* an additional gradient term

$$-[\lambda - \eta C(x)]\partial C(x)/\partial x \qquad (2)$$

is added for each constraint. The conventional approach is to minimize in iteration *"k" L(x)* for a fixed value of the Lagrange multiplier λ^k, then update it for the next step of iterations from the value λ^k to a new one λ^{k+1} using

$$\lambda^{k+1} - \lambda^k = \eta C(x) \qquad (3)$$

and repeat the minimization of *L(x)* with this new value. This is similar to converging the density for some specific set of atomic positions, changing these positions then reconverging. This is what is done with **PORT** in **mini,** and many other DFT codes.

The speed of the convergence with **PORT** is controlled mainly by the product of the number of eigenvalue clusters of the dielectric matrix and the force matrix (with a weaker dependence on the width of the clusters). The **MSR1a** algorithm converges (mixes) the atomic positions and density at the same time. Its speed is controlled by the number (and width) of the eigenvalue clusters of the joint dielectric and force matrix, which is typically much smaller than the product. In this code, similar to **MSR1a**, the Lagrangian variable is treated as a variable (i.e. similar to a single density value), $\eta C(x)$ as a residue (gradient) and they are mixed at the same time as everything else. Since this is only one (and perhaps in the future a few) extra variable it makes almost no difference to the speed of convergence.

A little additional care is needed with the constraints *C(x)*. The simplest form is a linear one such as x-0.5 but this has disadvantages as it can be very large far from the constraint and end up dominating the mixing. It is not hard to show that without changing the form, one can multiply by some function to damp the value far from the solution. A few different forms are optional.

The algorithm performs this simultaneous mixing of the density, atomic positions and the Lagrange multiplier. In addition to the conventional forces, it will also print out the pseudo-forces which are the combination of the conventional ones and the additional gradient term from equation (2). The convergence is monitored by the combination of these pseudo-forces and the movement, and when both are small enough it will stop and switch to a final MSR1 step. (Currently the deviation of the constraint is not used as a termination condition.) The constraint will, in general, not be exactly satisfied – but this is fine. If needed an additional iteration can be done with it satisfied, or the magnitude of the constantη, the strength of the penalty function term (**FCONST**, typical value 0.1) increased.

The algorithm will stay fairly close to the Born Oppenheimer surface (BOS), and will approach the constraint as it converges. However, it may wander away from the constraint for a while and appear to be doing strange things. These are almost certainly not bugs or problems and user intervention is probably inappropriate. For instance, at some stage the algorithm may have wandered a bit too far from the BOS and be adjusting the density variable and/or other atomic positions, and moving further from the constraint, or appear to be oscillating. This is fine.

One final comment concerns the constant η, the strength of the penalty function term (the input term **FCONST**, typical value 0.1). Far from the solution damping helps, near the solution it can be increased but it should never exceed 1.0 as this can lead to divergence. It may be that a smaller value is needed to avoid Ghost Bands for poorly conditioned problems, but this will take longer (more iterations). Alternatively use the Lagrangian method without augmentation.

IMPORTANT. The indications are that the **MSR1a** algorithm, and to a weaker extent **MSECa**, both will tend towards ***global*** fixed-points of the density and atomic positions only weakly guided by the forces. It can happen (with coding errors) that the atomic positions match the constraints, the density is converged by the pseudo-forces are large. Oddly enough this is "correct", but there is probably a coding error as the pseudo-forces should be small. If this occurs please let me know.

WARNING: Be aware that you are not calculating a trajectory on the "potential energy surface". Instead you are calculating a potential energy surface with one constraint. For instance, there is ***no reason*** that a pincer calculation and one with a single atom constraint will lead to identical results

except at a local minimum or a Transition State where all forces should be zero. Please be careful how you think about this.

Options

Following these lines are a number of options, which matter. More than one can be included, the latest taking precedence in case of conflicts. The first set control the mode used, with **WALK** and **TRUST 20** the default.

WALK

This is the default fail-safe method, which is slow and might not converge. It increments the effective force applied as needed until the final position is reached. A value of 0.1 for **FCONST** is reasonable, larger values such as 0.25 will help cases which won't converge. If you are restarting from a converged calculation close to the maximum, a larger value of **SCALE** might help.

TRUST XX

This applies an upper limit of XX mau for the displacement in a given step, default 20. It is useful in **WALK** and other modes to avoid too large movements. It is on by default.

AUGMENT

This applies an Augmented Lagrangian where the actual Lagrangian constant is mixed, rather than being incremented as is done for optimization problems. Some care is needed with **FCONST**, and I recommend a value of 0.05-0.1 when the starting point is far from the desired end point, a larger number such as 0.5 when it is close. A value of 0.9 or large will produce a warning as for mathematical reasons (over-relaxation) they are likely to lead to problems. Negative values can be used, they slow the algorithm down (under-relaxation).

LAGRANGIAN

This implements a simple Lagrangian, as against an Augmented Lagrangian. It works, and is often more gentle, albeit slower.

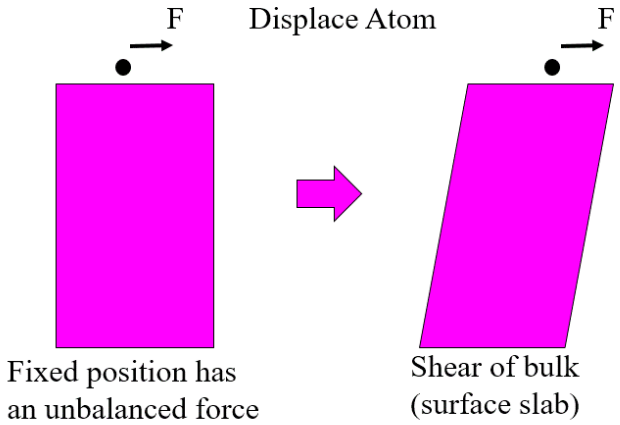
RESTART

This is needed if you are restarting from a previous calculation, and want to set the value of the Lagrange multiplier so the pseudo-forces are small initially. For instance, you might be tightening the constraint and moving from the previous value to a new one.

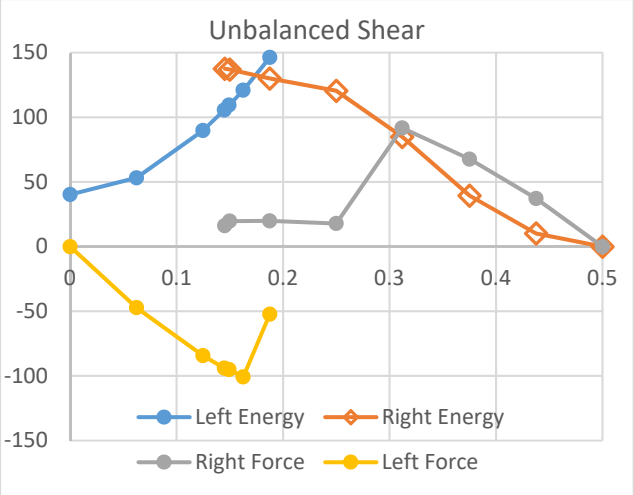
MIN

This is the default. It will typically start with a fairly small pseudo-force, and then increase it. It is needed, for instance, if the constraint is applied to the atomic densities from dstart.

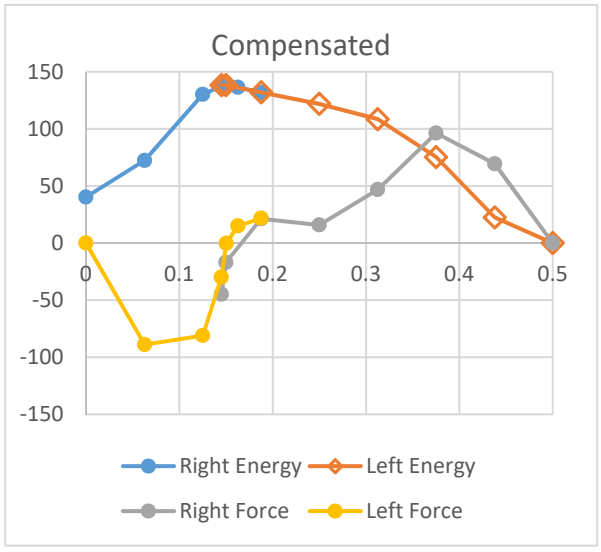
Additional Corrections



In addition to the above, some options for corrections are used and are ***very important***. To explain this, consider that the problem of interest is to find the transition state for an atom on a surface by moving it as illustrated to the right. Because moving an atom away from equilibrium is equivalent to applying a Force to it, this will have the effect of shearing the underlying bulk. Because of this shear, the behavior when you start from an "initial state" and move towards the "final state" will be different from going the other way – the system will have hysteresis. This is shown for a specific example in the Figure below and to the left. The blue and orange show the energies starting from x=0 and x=0.5 for a target atom, and the yellow and grey the corresponding Forces after full minimization of all other co-ordinates.



In order to avoid this the unbalanced forces and/or moments have to be cancelled. Shown to the right is cancellation of the moment by applying a simple shear to oppose it. This eliminates the hysteresis except very close to the Transition State, when an interesting phenomenon occurs. The actual Transition State appears to be a fixed-point attractor, so if by chance the end point is very close, the algorithm can terminate very close to it. This is not guaranteed to always occur, but has a reasonably high probability.



Corrections that are currently in the code are:

MOMENTX

Cancels moments normal to the x-axis, taking an origin of x=0. If an additional line **DISPLACE** is present then an origin of x=0.5 will be used.

MOMENTY or **MOMENTZ**

The same, for the y-axis or z-axis.

Note Full rotation compensation has not been implemented yet, it would not be hard to add if needed.

DRIFTX

Cancels any net applied force along the x-axis. *This has not been tested.*

DRIFTY or **DRIFTZ**

The same for the y-axis or z-axis. Multiple drift corrections are possible.

An example of a complete "working" **case.constraint** file is:

1002 1 0.1
0.125
/home/ldm/Mixing/NEB/Model3c/Struct1a/Struct1a.struct
1
1 0.0 -0.5 0.0
MOMENTY

How to Use

Exactly how to use the code to find Transition States (TS) and in other cases will almost certainly evolve with time, so these ideas are suggestions only. For TS I will assume that the Initial and Final states have already been determined, and a key atom that changes has been identified and the relevant co-ordinate determined. A method is:

1) Setup a number of cases stepping from the Initial to the Final. Not too many, not too few; probably 8 is enough.
2) Calculate from the end points towards the middle (e.g. 4 from the Initial, 4 from the Final) with an appropriate correction. I would probably do these in parallel, although an argument could be made for doing them sequentially. In some cases a displacement of a target atom may be best, in others a pincer movement to change the distance between two atoms.
3) Pick, if possible, options to correct for unbalanced forces.
4) Check for hysteresis.
5) Plot the (converged) energies and the forces on the target atom.
6) Create a scatter plot of the energy and (perhaps) forces during the run(s). This sometimes shows up features.
7) Create a scatter plot of the x,y (or x,z; y,z) coordinates during the run(s) – 3D would be nice.
8) Hopefully the TS will now be clear. Bracket somehow moving closer – and there is a strong chance that you will hit the TS because it can be a weak attractor.
9) Perhaps (very perhaps) from a point close to the TS use **MSECa**. It might well converge (it should) to the TS. **MSR1a** probably won't, although you might be lucky!
10) In the end, do a "mini-phonon" for the target atom alone (without correction probably) about the TS – I don't think a full phonon of all atoms is needed.

WARNING: Be aware that you are not calculating a trajectory on the "potential energy surface". Instead you are calculating a potential energy surface with one constraint. For instance, there is ***no reason*** that a pincer calculation and one with a single atom constraint will lead to identical results ***except*** at a local minimum or a Transition State where all forces should be zero. Please be careful how you think about this.

Other Comments

a. There may be something odd/wrong with using a displacement which is not just x, y or z only. I have not tested a case to know exactly.
b. Compensating for rotation might be relevant.
c. Only tested to date for cases with inversion symmetry; drift may be an issue as the constraints may break (or conflict with) the drift correction already in **MSR1a**.

Monitor Terms (for grep)

Several terms can be used to monitor the progress and look for problems. Some are only active if **VERBOSE** is included in case.inm. Most of these are after "**User defined constraints detected**" in case.scfm

Discrepancy for Atom	Estimate of distance/gradient to constraint
Vector Residue	Estimate of RMS distance for all relevant atoms
Raw Force Components	Relevant Forces
:**Lagrange Multiplier**	Value of the Lagrange Multiplier, worth monitoring

VERBOSE ONLY

Mixed Value and Residue	Terms being used in the mixing
Next Lagrangian	Value after mixing, that will be used in the next iteration

SPECIFIC Modes

Walk Mode Only

:LAG XXX	Information about how Lagrangian term is increasing or decreasing
Change Vector	Information about distance from constraint

Augmented Lagrangian

Unit Check	Sanity check on internal parameters

Additional Monitoring

It may be useful to use a script such as:

```
#!/bin/bash
grep :FGL$1 *.scf | cut -c 1-7,32-78 | tail -20 > tmp_tmp1
grep :FGC$1 *.scf | cut -c 1-7,32-78 | tail -20 > tmp_tmp2
paste tmp_tmp1 tmp_tmp2
rm tmp_tmp1 tmp_tmp2
```

to monitor both the pseudo-forces for a target atom, and the non self-consistent forces.